REMARKS

Claims 1-36 are pending in the application.

Claims 1-36 are rejected.

Claims 1, 7, and 24 have been amended to correct informalities.

Claims 2, 3, and 4 are amended to be consistent with the corresponding claims from which they depend.

The Applicants respectfully assert that the amendments to Claims 1, 2, 3, 4, 7, and 24 and incorporated by reference in any claims depending therefrom, are not narrowing amendments made for a reason related to the statutory requirements for a patent that will give rise to prosecution history estoppel. *See Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co.*, 122 S. Ct. 1831, 1839-40, 62 U.S.P.Q.2d 1705, 1711-12 (2002); 234 F.3d 555, 566, 56 U.S.P.Q.2d 1865, 1870 (Fed. Cir. 2001).

I. REJECTION UNDER 35 U.S.C. § 102(b)

The Examiner rejected Claims 1-5, 7, 9-13, 15-21, 22-28, and 32-36 under *35 U.S.C. §102(b)* as being anticipated by U.S. Patent No. 5,596,739 to *Kane et al.* (hereafter "*Kane*").

For a reference to anticipate a claimed invention, the reference must disclose every aspect of the claimed invention. *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631, 2 U.S.P.Q.2d 1051, 1053 (Fed. Cir. 1987). The identical invention must be shown in as complete detail as is contained in the claim. *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 U.S.P.Q.2d 1913, 1920 (Fed. Cir. 1989).

In general, *Kane* is disclosing the concept of memory protection via segment descriptors and process privilege rings (levels). See *Kane*, column 6, line 60. Segment and page protection as well as user privilege domains and partitioning of memory is not

what the present invention is disclosing. The present invention deals with an additional realm of stack memory protection.

The way an operating system (OS) traditionally manages memory is by using page protection bits to classify memory as not accessible, or accessible by one or more privilege classifications. The present invention discloses a special page protection that limits access to specific instructions under a single privilege domain or execution context. For instance, assume a normal load instruction that can load from memory is called LWZ. The present invention recites generating a new load (stack) instruction LWZSTK such that within a single program flow and a single privilege domain LWZ would be allowed to access memory classified as non-stack memory and LWZSTK would be allowed to access memory classified as stack memory by the OS. Likewise, an LWZ instruction is blocked from stack memory and an LWZSTK instruction is blocked from non-stack memory. This allows granular use-specific (instruction specific) classification and protection of memory by a single privilege/protection domain.

The present invention addresses the real problem with the IA64 processor register stack engine. This is a register stack engine that is transparent to the OS; it independently loads and stores data from/to the register memory stack outside of normal instruction execution. However, with the IA64 the OS still has to allocate and manage memory for this "additional" stack while also allocating and managing the traditional stack memory used by normal program instruction execution. The present invention applies additional stack memory classifications to this scenario such that the OS did not have to worry about stack boundaries of normal program instruction execution stack references overflowing into register stack memory and vice versa.

The present invention addresses the problem of an operating system's difficulty in setting up and protecting stack memory regions. There are numerous memory stacks that the OS must manage, e.g., interrupt stacks, kernel thread stacks, user stacks, signal stacks, pthread stacks, etc. In addition, future processors may conceivably compound the complexity by introducing multiple stacks per execution context. One of the classic

issues in managing a memory stack is protecting against overrun/underrun. Traditionally, this has been solved by allocating "red zones", or protected pages to hopefully cause faults when an overrun or underrun occurs. The problem (which may compound in the future) is the waste of pages (also compounded by your selected page size) required to implement these "red zones" for every stack in the system. Another problem is that the red zones do not always work. For example, a stack pointer may be decremented by a value equal to or greater than page size, and effectively skip right over the "red zone". The present invention creates a <u>new memory protection classification</u> specific for "stack memory". The invention includes a CPU design/implementation enhancement, <u>requiring support from operating systems and compilers</u>. Provided that a CPU supported this <u>new protection attribute</u>, i.e., page table entry attribute, the OS could map each page of a "stack" (both present and future) with this new protection. Compilers would generate <u>specific forms of load/store instructions when saving or restoring to/from stacks</u>. These <u>new instruction forms</u> would result in the memory references requiring the "stack protection" attributes to succeed or raise a fault when they do not succeed. Likewise, for "future" stack uses, some potentially transparent to the compiler or programmer, all references generated internally by the CPU would be such that the appropriate "stack protection" attributes were set.

Therefore, the stacked memory is now further protected in that errant "normal" loads and stores to "stack" memory cause faults, just as errant "stack" loads and stores cause faults, further with the present invention there are no more "red zones" required to be setup, wasted, managed by the operating system.

The present invention creates a <u>new page attribute</u>, i.e., part of the page table and the translation look aside buffer (TLB) entries that already have information line (read-only, read-write, kernel privilege, user privilege, etc.), that mark memory pages as "stack memory". Considering the particular IA64 architecture, there could be an attribute to mark memory pages as "register stack" memory. In conjunction, the processor would be required to use a specific bus transaction (<u>new instruction</u>), different from normal

load/store instructions, to read or write to memory pages containing this new page attribute. Likewise, normal load/store instructions would not be allowed to pages with the new attribute and new "stack" load/store instructions would not be allowed to memory pages without this new page attribute. An advantage to the OS, with the present invention, is that it can locate these new separate stack regions anywhere desired in the address space, adjacent, growing together, growing apart, etc., with the knowledge that the stack memory regions are protected from each other and from normal memory operations.

Claim 1 recites a method for stack memory protection comprising 4 steps. In step 1, new memory page attributes are generated for a page table used to manage memory, wherein each of the new memory page attributes identify a block of memory as a new class of memory, and each of the new memory page attributes are generated by a corresponding new load/store instruction. In step 2, an operating system or a processor assigns a selected one of the new memory page attributes to a selected block of memory used as a new class of memory corresponding to the selected new memory page attribute. In step 3, normal load/stores to a memory block having one of said new memory page attributes are blocked; and in step 4, a first new load/store to a memory block with one of the new memory page attributes not corresponding to the first new load/store is blocked.

The Examiner states that *Kane* teaches step 1 of Claim 1 and cites *Kane*, Fig. 1 and column 1, lines 40-46 and 56-59. *Kane*, in column 1, lines 40-46 is describing the concept of memory protection using privilege levels. In lines 56-59, *Kane* is describing a portion of his invention including a memory control unit for a microprocessor that can run multiple tasks wherein the control unit has segment registers for directing memory access requests to specified segments in memory associated with the memory control unit. In particular, the microprocessor assigns the privilege levels to the segments in memory and assigns privilege levels to the multiple tasks as a form of memory protection. Nowhere in this citation does *Kane* teach or suggest that new memory page

attributes are generated identifying a block of memory as a new class of memory wherein each new memory page attribute is generated by a new load/store instruction.

The Examiner states that *Kane* teaches step 2 of Claim 1 and again recites column 1, lines 56-59. In this recitation, *Kane* states that the processor assigns privilege levels to the segments in the memory and assigns privilege levels to the multiple tasks. These are normal privilege types of protection. In Claim 2, the OS or processor assigns a new memory page attribute to a selected block of memory used as a new class of memory. This is done in addition to any normal privilege assigned as outlined by *Kane*. The present invention does not prevent any normal assignment of privilege from being used; rather, the present invention generates new attributes that define a new class of memory that is protected from access by normal instructions. However, the new class of memory may continue to use standard privilege within the class. The assignment by *Kane* has nothing to do with assigning a new attribute of the present invention to a selected block of memory thereby creating a new class of memory accessible only by new in load/store instructions. Nowhere in this recitation does *Kane* teach or suggest step 2 of Claim 1.

The Examiner states that *Kane* teaches step 3 of Claim 1 and cites *Kane*, column 2, lines 8-11. In this recitation, *Kane* states that "the protection circuit permits the memory control unit to load a segment register to identify the specified memory segment only when the specified memory segment is accessible by the task." *Kane* is describing how his memory control unit uses a programmable protection register to allow or deny identification of a segment with a particular privilege when access to the segment is attempted by a task. The teachings of *Kane* are transparent to the present invention. The present invention generates new memory page attributes. When the OS or processor assigns a new attribute to a block of memory, a new class of memory is created which is accessed only by new load/store instructions. A task (with a privilege level), according to *Kane*, could execute one of the new load/store instructions and write a new attribute to a block of memory creating the new class of memory accessible only by the new

load/store instructions. Likewise, segments (memory) of *Kane* (with particular privileges) could be identified in the block of memory using the same new load/store instructions. In this manner, the present invention is operable with the invention of *Kane*. However the present invention is not anticipated by *Kane* as *Kane* does not teach or suggest generating a new memory page attribute, which when assigned to a block of memory, creates a new class of memory accessible only with new load/store instructions, wherein normal load/store instructions are blocked from the block of memory with the assigned new attribute.

The Examiner states that *Kane* teaches step 4 of Claim 1 and cites *Kane*, column 2, lines 2-11. In this recitation, *Kane* states that "The programmable protection register is indexed by an index formed by combining a privilege level of the task requesting the specified memory segment and a privilege level of the specified memory segment. The protection register outputs at least one value responsive to the index that indicates whether the memory segment is accessible by the task. The protection circuit permits the memory control unit to load a segment register to identify the specified memory segment only when the specified memory segment is accessible by the task." *Kane* is describing how his memory control unit uses a programmable protection register and an index to generate and output a value indicating whether a memory segment is accessible by the task. The teachings of *Kane* are transparent to the present invention. The present invention generates new memory page attributes. When the OS or processor assigns a new attribute to a block of memory, a new class of memory is created which is accessed only by new load/store instructions. A task (with a privilege level), according to *Kane*, could execute one of the <u>new load/store instructions (of the present invention)</u> and write a <u>new attribute to a block of memory</u> creating <u>the new class of memory accessible</u> only by the new load/store instructions. Likewise segments (memory) of *Kane* (with particular privileges) could be identified in the block of memory using the same new load/store instructions. In this manner, the present invention is operable with the invention of *Kane*. However, the present invention is not anticipated by *Kane* as *Kane* does not teach or suggest generating <u>a new memory page attribute</u> that, when assigned

to a block of memory, creates a new class of memory accessible only with new load/store instructions, wherein the new load/store instructions are blocked from the block of memory without the assigned new attribute.

Therefore, the Applicants respectfully assert that the rejection of Claim 1 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments.

Amended Claim 2 is dependent from Claim 1 and contains all the limitations of Claim 1. Claim 2 recites the limitation that the new class of memory includes stack memory with corresponding stack memory load/store instructions. Therefore, Claim 2 recites the invention of Claim 1, wherein a new memory page attribute defines a block of memory as "stack" memory with corresponding new "stack" memory load/store instructions. In a particular case, a block of memory (block 1) is assigned a particular (new) memory page attribute (e.g., "stack 1") which defines block 1 as a new class of memory, "stack memory 1." A corresponding new load/store instruction (e.g., "load stack 1") is created that is used by the OS or the processor when it wants to read from "stack memory 1." This is an important distinction from the invention of *Kane* which does not assign a new attribute to a block of memory wherein a new instruction is generated for accessing the block with the new attribute. *Kane* allows normal instructions to attempt to access any of his memory and thus requires complex operations (his memory control unit) to determine if the instruction has the correct privilege to access the memory. The present invention eliminates this complexity by generating a new instruction for the block of memory with the new attribute and thus the present invention does not have to be involved in complex determination of privilege levels as recited by *Kane* for the new class of memory. The present invention uses the generation of new classes of memory with corresponding new attributes to bypass the complexity of *Kane*, in particular for the class of memory defined as "stack memory." This reduces the overhead of the processor and the OS for managing memory, in particular, stack memory. Therefore, the Applicants respectfully assert that the rejection of Claim 2 under *35 U.S.C. §102(b)* as being

anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 1.

Claim 3 has been amended to more clearly define the limitation of generating a new load/store instruction as introduced in Claim 1. Claim 3 is dependent from Claim 2 and contains all the limitations of Claims 1 and 2. Amended Claim 3 adds the limitation that a first error condition that is generated whenever normal load/stores are attempted to the stack memory of Claim 2 that has a first or a second stack memory attribute. Again, the invention of amended Claim 3 eliminates the complexity of *Kane* for new classes of memory, in particular, stack memory. Claim 3 recites that when a new attribute (first or second stack memory attribute) is assigned to a block of memory (creating first or second stack memory) and corresponding new load/store instruction (e.g., stack load/store 1 and stack load/store 2) for the block of memory with the first or second stack memory attribute, then if a <u>normal load/store</u> is attempted (by the OS or the processor) as opposed to a stack load/store 1 (first stack memory) or a stack load/store 2 (second stack memory) then a first error condition is generated. *Kane* goes through a complex series of operations to determine if the task has the privilege to access a particular segment of memory. In the present invention, a task desiring access to a block of memory (e.g., first stack memory) with a new assigned attribute (first stack memory attribute) would need to execute a stack load/store 1. If the task attempted a normal load/store to the first stack memory, a first error condition would be generated. For the present invention, accesses are determined at the <u>instruction level</u> relieving the processor or OS from having to determine through the complexity described by *Kane* whether a particular block of memory may be accessed. Therefore, the Applicants respectfully assert that the rejection of amended Claim 3 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 2.

Claim 4 is amended to correct an antecedent basis problem more clearly defining the limitations. Amended Claim 4 is dependent from Claim 2 and contains all the limitations of Claim 2. Claim 4 adds the limitation that a second error condition that is

generated whenever stack memory load/stores are attempted to the memory of Claim 2 that does not have a first or a second stack memory attribute. Again, the invention of Claim 4 eliminates the complexity of *Kane* for new classes of memory, in particular, stack memory. Claim 4 recites that when a new attribute (first or second stack memory attribute) is assigned to a block of memory (creating first or second stack memory) and corresponding new load/store instruction (e.g., stack load/store 1 and stack load/store 2) for the block of memory with the first or second stack memory attribute, then if a <u>normal load/store</u> is attempted (by the OS or the processor) as opposed to a stack load/store 1 (first stack memory) or a stack load/store 2 (second stack memory), then a first error condition is generated. *Kane* goes through a complex series of operations to determine if the <u>task</u> has the privilege to access a particular <u>segment of memory</u>. In the present invention, a task desiring access to a block of memory (e.g., first stack memory) with a new assigned attribute (first stack memory attribute) would need to execute a stack load/store 1. If the task attempted a normal load/store to the first stack memory, a first error condition would be generated. For the present invention, accesses are determined at the <u>instruction level</u> relieving the processor or OS from having to determine through the complexity described by *Kane* whether a particular block of memory may be accessed. Therefore, the Applicants respectfully assert that the rejection of amended Claim 4 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 2.

Claim 5 is dependent from Claim 2 and contains all the limitations of Claim 2. Claim 5 adds the limitation of a third error condition that is generated whenever a stack memory load/store for a first memory stack is attempted to a second memory stack, said third error condition also generated if a stack memory load/store for said second memory stack is attempted to said first memory stack. From Claim 2, a first memory stack is formed (by assigning a first stack memory attribute to a first block of memory) and a second memory stack is formed (by assigning a second stack memory attribute to a second block of memory). In Claim 5, the third error condition is generated whenever a stack memory load/store for the first memory stack (e.g., stack load/store 1) is

attempted to the second block of memory with the second stack memory attribute. Likewise, Claim 5 recites that the third error is generated whenever a stack memory load/store for the second memory stack (e.g., stack load/store 2) is attempted to the first block of memory with the first stack memory attribute. *Kane* goes through a complex series of operations to determine if the task has the privilege to access a particular segment of memory. In the present invention, a task desiring access to a block of memory (e.g., first stack memory) with a new assigned attribute (first stack memory attribute) would need to execute a stack load/store 1 according to embodiments of the present invention. If the task attempted a load/store 2 to the first stack memory, a first error condition would be generated. For the present invention, accesses are determined at the instruction level relieving the processor or OS from having to determine through the complexity described by *Kane* whether a particular block of memory may be accessed. Therefore, the Applicants respectfully assert that the rejection of amended Claim 5 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 2.

Claim 7 is dependent from Claim 5 and contains all the limitations of Claim 5. Claim 7 adds the limitation that the first memory stack is a processor stack used by a processor to load and store hardware register contents during program execution, wherein the processor stack is transparent to a programmer or a compiler. The Examiner states that *Kane* teaches Claim 7 and cites *Kane*, column 8, lines 3-17. This recitation of *Kane* discusses a method that is similar to the method of FIG 4 (of *Kane*) wherein the segment registers are each loaded with an index, respectively, that identifies a particular descriptor in either the global descriptor table or the local descriptor table. The identified descriptors are loaded into the corresponding descriptor registers. The descriptor registers identify different memory segments of the user memory. Nowhere in this recitation does *Kane* teach or describe that a first memory stack that is formed by assigning a stack memory attribute to a block of memory with a corresponding first stack memory load/store instruction is a processor stack used by the processor to load and store hardware register contents. Further, *Kane* does not teach or describe that the processor

stack, so formed, is transparent to a programmer or compiler using the processor. The present invention and in particular the invention of Claim 7 is transparent to a programmer or compiler because a programmer writing code, for a processor with the stack memory attributes of the present invention, would have the processor assign the stack memory attributes. Thus, using the invention of Claim 7, stack memory may be assigned anywhere in memory without creating wasted protected pages or having data corrupted by stack memory overruns or under runs. Using the invention of Claim 7, the OS no longer needs to allocate specific space in memory as stack memory and likewise does not have to estimate how much memory is needed for program stacks and processor stacks as is presently required in the IA64 register stack architecture. Therefore, the Applicants respectfully assert that the rejection of Claim 7 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 5.

Claim 9 is dependent from Claim 5 and contains all the limitations of Claim 5. Claim 9 adds the limitation that the second memory stack is a program stack used by a programmer or a compiler in managing program flow. The Examiner states that *Kane* teaches Claim 9 and cites *Kane*, column 8, lines 25-32. This recitation of *Kane* discusses the method wherein memory segments may be accessed by referencing the corresponding segment registers. Nowhere in this recitation does *Kane* teach or describe that a second memory stack that is formed by assigning a stack memory attribute to a block of memory with a corresponding second stack memory load/store instruction is a program stack used by the processor to manage program flow. The Applicants have shown that *Kane* does not teach or suggest the invention of Claim 5 and thus *Kane* does not teach or suggest the invention of Claim 9 further limiting the invention of Claim 5. Therefore, the Applicants respectfully assert that the rejection of Claim 9 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 5.

Claim 10 is directed to a processor using blocks of memory as stack memory and having stack memory protection circuitry having four elements. First, a stack memory attribute circuit operable to generate memory attributes associated with each memory block designated as a memory stack. Second, a page table attribute storage circuit operable to store and associate one of the stack memory attributes with a block of memory designated as stack memory. Third, a stack memory allocation circuit operable to identify a block of memory as a stack memory and associate the memory block with one of the stack memory attributes, wherein the stack memory attributes are stored in a memory page table. Fourth, a stack memory instruction execution circuit operable to decode load/store instructions to memory blocks and grant stack memory load and stores to memory blocks having a required stack memory attribute and not granting stack memory load and stores to memory blocks not having said required stack memory attribute.

The Examiner states that *Kane* teaches and describes the processor of Claim 10 with the first element, the stack memory attribute circuit, and cites *Kane*, Fig. 1 and column 1, lines 40-46 and 56-59. Fig. 1 of *Kane* is a functional block diagram of a processor with the memory protection circuit described in the invention of *Kane*. While the figure indicates a protection unit 180, it does not show a stack memory attribute circuit and thus in itself does not read on the invention of Claim 10.

In column 1, lines 40-46, *Kane* is describing the four privilege levels of the protection mechanism of a generic 486 microprocessor wherein each memory segment and each software routine is assigned a privilege level. In this protection mechanism of a "generic" 486 processor, the protection mechanism monitors memory accesses and implements rules related to those memory accesses wherein generally less privileged software routines are not allowed access to more privileged memory segments. Nowhere in this recitation is stack memory mentioned nor is a stack memory attribute circuit mentioned. *Kane* is describing the present art of memory protection that the invention of Claim 10 is improving upon.

In column 1, lines 56-59, *Kane* states that his memory control unit has segment registers for directing memory access requests to specified segments in memory associated with the memory control unit. The microprocessor of *Kane* further assigns privilege levels to the segments in the memory and assigns privilege levels to the multiple tasks. Nowhere in this recitation is stack memory mentioned nor is a stack memory attribute circuit mentioned. The Examiner fails to point out what he considers in this recitation of *Kane* is the stack memory attribute circuit as recited by the first element of Claim 10.

The Examiner states that *Kane* teaches and describes the processor of Claim 10 with the second element, the page table attribute storage circuit, and cites *Kane*, Fig. 2A, element 330 and column 5, lines 6-16. Fig. 2A of *Kane* illustrates the external memory 390 connected to the memory control unit 115 of a processor 100. The Examiner states that element 330 (global descriptor table) is the page table attribute storage circuit of Claim 10. Element 330 of *Kane* is a <u>table</u> in memory 390 and not a page table attribute storage circuit. In Claim 10 of the present invention, the stack memory attribute circuit (first element) generates memory attributes (associated with memory blocks designated as <u>stack memory</u>) and the page table attribute storage circuit (second element) stores and associates one of the stack memory attributes with a block of memory designated as stack memory. Element 330 of *Kane* is a table that in memory for holding descriptor values and is not a not a page table attribute storage circuit as recited in element two of Claim 10.

The Examiner states that *Kane* teaches and describes the processor of Claim 10 with the third element, the stack memory allocation circuit, and cites *Kane*, column 1, lines 56-59. The Examiner has already stated that he believes column 1, lines 56-59 teaches the stack memory attribute circuit that generates memory attributes. Now the Examiner states that the same description teaches the stack memory allocation circuit that identifies a block of memory as stack memory and associates the block of memory with one of the stack memory attributes.

In column 1, lines 56-59, *Kane* only states that his memory control unit has segment registers for directing memory access requests to specified segments in memory associated with the memory control unit. The microprocessor of *Kane* further assigns privilege levels to the segments in the memory and assigns privilege levels to the multiple tasks. Nowhere in this recitation is stack memory mentioned nor is a stack memory allocation circuit mentioned. The Examiner cannot have one element of *Kane* represent two distinct elements of the present invention.

The Examiner states that *Kane* teaches and describes the processor of Claim 10 with the fourth element, the stack memory instruction execution circuit that grants stack memory load and stores to memory blocks having a required stack memory attribute and not granting stack memory load and stores to memory blocks not having said required stack memory attribute, and cites *Kane*, column 3, lines 18-23, column 14, lines 36-42 and column 2, lines 2-11.

Kane, in column 3, lines 18-23, is describing units within a microprocessor that is suitable for practicing the invention of *Kane*. The Examiner is implying that one of the units in a standard microprocessor is the "stack memory instruction execution circuit" of Claim 10. Element four of Claim 10 is not describing any instruction execution unit; rather, element four of Claim 10 is a particular stack memory instruction execution circuit that grants stack memory loads and stores to memory blocks with required stack memory attributes and does not grant stack memory load and stores to memory blocks without the required stack memory attributes.

Kane, in column 14, lines 36-42 describes the protection level decoder 610 that generates signals on protection level fault lines 184 to indicate test results. *Kane*, in this recitation, further describes what types of tests may be performed that generate the protection level fault signals. Nowhere in this recitation does *Kane* teach or disclose the stack memory instruction execution circuit of Claim 10.

Kane, in column 2, lines 2-11, describes how his memory control unit uses at least one programmable protection register that is indexed by an index generated by combining the privilege level of task and the privilege level of a memory segment to determine if the memory segment is accessible by the task. Nowhere in this recitation does *Kane* teach or disclose the stack memory instruction execution circuit of Claim 10.

The Applicants have shown that *Kane* does not teach <u>all of the elements of Claim 10 which is required</u> to make a *prima facie* case of anticipation. Therefore, the Applicants respectfully assert that the rejection of Claim 10 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments.

Claim 11 is dependent from Claim 10 and contains all the limitations of Claim 10. Claim 11 adds the limitation that a first error condition is generated whenever normal load/stores are attempted to stack memory having a first or a second stack memory attribute. The Examiner states that *Kane* teaches and describes the invention of Claim 11 and cites *Kane*, column 13, line 66-column 14, line 3. The Applicants have shown that *Kane* does not teach or disclose the processor of Claim 10. Claim 11 is describing an error condition generated when normal load/stores are attempted to stack memory that has a first or a second stack memory attribute generated by the stack memory attribute circuit, wherein the stack memory was generated when a block of memory was assigned the first or second stack memory attributes by the stack memory allocation circuit. Further, the stack memory instruction execution circuit would have decoded the normal load/stores attempted to the stack memory and not granted access to the stack memory. While the invention of *Kane* generates protection level faults for a variety of conditions, *Kane* is not describing the processor of Claim 10 implementing the particular <u>stack memory protection</u> of the present invention with the limitation of Claim 11. *Kane* does not have instruction level decodes where load/store instructions check instruction checks for the presence or absence of a stack memory attribute. The present invention looks for the presence or absence of stack memory attributes at <u>the instruction level</u> regardless of the <u>task executing</u> the instruction. *Kane* matches task privilege to memory segment

privilege, wherein the privilege levels have hierarchy. The present invention may also use privilege; however, the present invention does not allow, under any privilege, stack memory load/stores to non-stack memory or normal load/stores to stack memory. Therefore, the Applicants respectfully assert that the rejection of Claim 11 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 10.

Claim 12 is dependent from Claim 10 and contains all the limitations of Claim 10. Claim 12 adds the limitation that a first error condition is generated whenever the stack memory load/stores are attempted to memory not having a stack memory attribute. The Examiner states that *Kane* teaches and describes the invention of Claim 12 and cites *Kane*, column 13, line 66-column 14, line 3. The Applicants have shown that *Kane* does not teach or disclose the processor of Claim 10. Claim 12 is describing an error condition generated when stack memory load/stores are attempted to memory that does not have a stack memory attribute generated by the stack memory attribute circuit. Further, the stack memory instruction execution circuit would have decoded the stack memory load/stores attempted to the non-stack memory and not granted access to the normal memory. Error conditions are generated only when a normal load/store is attempted to stack memory (with a stack memory attribute) and when a stack memory load/store is attempted to non-stack memory (without a stack memory attribute). *Kane* does not have instruction level decodes where load/store instructions check for the presence or absence of a stack memory attribute. The present invention looks for the presence or absence of stack memory attributes at the instruction level regardless of the task executing the instruction. *Kane* matches task privilege to memory segment privilege, wherein the privilege levels have hierarchy. The present invention may also use privilege; however, the present invention does not allow, under any privilege, stack memory load/stores to non-stack memory or normal load/stores to stack memory. Therefore, the Applicants respectfully assert that the rejection of Claim 12 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 10.

Claim 13 is dependent from Claim 10 and contains all the limitations of Claim 10. Claim 13 adds the limitation that a third error condition is generated whenever a stack memory load/store for a first memory stack is attempted to a second memory stack and the third error condition also is generated if a stack memory load/store for said second memory stack is attempted to said first memory stack. The Examiner states that *Kane* teaches and describes the invention of Claim 13 and cites *Kane*, column 13, line 66-column 14, line 3. The Applicants have shown that *Kane* does not teach or disclose the processor of Claim 10. Claim 13 is reciting a hierarchy within the stack memory load/store instructions of the present invention. In Claim 13, a first stack memory has a particular first stack memory load/store instruction. The first stack memory would be assigned a stack memory attribute indicating that it is a particular memory stack, the first memory stack. Likewise, the second stack memory would be assigned a stack memory attribute indicating that it is a particular memory stack, the second memory stack. In the invention of Claim 13, stack memory load/store instructions are particularly directed to specific stack memory and a third error condition is generated if a stack memory load/store for a first memory stack is directed to a second memory stack and visa-versa. *Kane* does not have instruction level decodes where load/store instructions check for the presence or absence of a stack memory attribute. The present invention looks for the presence or absence of stack memory attributes at <u>the instruction level</u> regardless of the <u>task executing</u> the instruction. *Kane* matches <u>task privilege to memory segment privilege</u>, wherein the privilege levels have hierarchy. The present invention may also use privilege; however, the present invention does not allow, <u>under any privilege</u>, stack memory load/stores to non-stack memory or normal load/stores to stack memory. Therefore, the Applicants respectfully assert that the rejection of Claim 13 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 10.

Claim 15 is dependent from Claim 13 and contains all the limitations of Claim 13. Claim 15 adds the limitation that the first memory stack is a processor stack used by a processor to load and store hardware register contents during program execution,

wherein the processor stack is transparent to a programmer or a compiler. The Examiner states that *Kane* teaches Claim 15 and cites *Kane*, column 8, lines 3-32. This recitation of *Kane* discusses a method that is similar to the method of FIG. 4 (of *Kane*) wherein the segment registers are each loaded with an index, respectively, that identifies a particular descriptor in either the global descriptor table or the local descriptor table. The identified descriptors are loaded into the corresponding descriptor registers. The descriptor registers identify different memory segments of the user memory. Nowhere in this recitation does *Kane* teach or describe that a first memory stack that is formed by assigning a stack memory attribute to a block of memory with a corresponding first stack memory load/store instruction is a processor stack used by the processor to load and store hardware register contents. Further, *Kane* does not teach or describe that the processor stack so formed is transparent to a programmer or compiler using the processor. The present invention and in particular the invention of Claim 15 is transparent to a programmer or compiler because a programmer writing code, for a processor with the stack memory attributes of the present invention, would have the processor assign the stack memory attributes. Thus, using the invention of Claim 15, stack memory may be assigned anywhere in memory without creating wasted protected pages or having data corrupted by stack memory overruns or underruns. Using the invention of Claim 15, the OS no longer needs to allocate specific space in memory as stack memory and likewise does not have to estimate how much memory is needed for program stacks and processor stacks as is presently required in the IA64 register stack architecture. Therefore, the Applicants respectfully assert that the rejection of Claim 15 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 13.

Claim 16 is dependent from Claim 13 and contains all the limitations of Claim 13. Claim 16 adds the limitation that the second memory stack is a program stack used by a programmer or a compiler in managing program flow. The Examiner states that *Kane* teaches Claim 16 and cites *Kane*, column 8, lines 25-32. This recitation of *Kane* discusses the method wherein memory segments may be accessed by referencing the

corresponding segment registers. Nowhere in this recitation does *Kane* teach or describe that a second memory stack that is formed by assigning a stack memory attribute to a block of memory with a corresponding second stack memory load/store instruction is a program stack used by the processor to manage program flow. The Applicants have shown that *Kane* does not teach or suggest the invention of Claim 13 and thus *Kane* does not teach or describe the invention of Claim 16 further limiting the invention of Claim 13. Therefore, the Applicants respectfully assert that the rejection of Claim 16 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed by the above arguments and for the same reasons as Claim 13.

Claim 17 is directed to a data processing system having a CPU, RAM, ROM, I/O adapter, and a bus system coupling the CPU, RAM, ROM and I/O adapter, wherein the CPU further has stack memory protection circuitry having the elements recited in Claim 10 of the present invention. The Examiner states that *Kane* teaches and describes the RAM, ROM, I/O adapter, bus system, and the CPU with the stack memory protection circuitry having the elements recited in Claim 10. The key to Claim 17 is the elements of the data processing system with the addition of the novel stack memory protection circuitry. The Examiner rejected Claim 17 for the same reasons as Claim 10 except for his rejection of the elements comprising the RAM, ROM, I/O adapter, and the bus system. Therefore, the Applicants respectfully assert that the rejection of Claim 17 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claim 10.

Claim 18 is dependent from Claim 17 and contains all the limitations of Claim 17. Claim 18 adds the same limitation to Claim 17 as Claim 11 adds to Claim 10. The Examiner has rejected Claim 18 for the same reasons as he rejected Claim 11. Therefore, the Applicants respectfully assert that the rejection of Claim 18 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 10 and 11.

Claim 19 is dependent from Claim 17 and contains all the limitations of Claim 17. Claim 19 adds the same limitation to Claim 17 as Claim 12 adds to Claim 10. The Examiner has rejected Claim 19 for the same reasons as he rejected Claim 12. Therefore, the Applicants respectfully assert that the rejection of Claim 19 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 10 and 12.

Claim 20 is dependent from Claim 17 and contains all the limitations of Claim 17. Claim 20 adds the same limitation to Claim 17 as Claim 13 adds to Claim 10. The Examiner has rejected Claim 20 for the same reasons as he rejected Claim 13. Therefore, the Applicants respectfully assert that the rejection of Claim 20 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 10 and 13.

Claim 22 is dependent from Claim 20 and contains all the limitations of Claim 20. Claim 22 adds the same limitation to Claim 20 as Claim 15 adds to Claim 13. The Examiner has rejected Claim 22 for the same reasons as he rejected Claim 15. Therefore, the Applicants respectfully assert that the rejection of Claim 22 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 10 and 15.

Claim 23 is dependent from Claim 20 and contains all the limitations of Claim 20. Claim 23 adds the same limitation to Claim 20 as Claim 16 adds to Claim 13. The Examiner has rejected Claim 23 for the same reasons as he rejected Claim 16. Therefore, the Applicants respectfully assert that the rejection of Claim 22 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 10 and 16.

Claim 24 is an independent claim directed to a computer program product that implements the method steps of Claim 1. The Examiner has rejected Claim 24 for the same reasons as he rejected Claim 1. Therefore, the Applicants respectfully assert that the rejection of Claim 24 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claim 1.

Claim 25 is dependent from Claim 24 and contains all the limitations of Claim 24. Claim 25 adds the same limitation to Claim 24 as Claim 2 adds to Claim 1. The

Examiner has rejected Claim 25 for the same reasons as he rejected Claim 2. Therefore, the Applicants respectfully assert that the rejection of Claim 25 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 1 and 2.

Claim 26 is dependent from Claim 24 and contains all the limitations of Claim 24. Claim 26 adds the same limitation to Claim 24 as Claim 3 adds to Claim 1. The Examiner has rejected Claim 26 for the same reasons as he rejected Claim 3. Therefore, the Applicants respectfully assert that the rejection of Claim 26 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 1 and 3.

Claim 27 is dependent from Claim 24 and contains all the limitations of Claim 24. Claim 27 adds the same limitation to Claim 24 as Claim 4 adds to Claim 1. The Examiner has rejected Claim 27 for the same reasons as he rejected Claim 4. Therefore, the Applicants respectfully assert that the rejection of Claim 27 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 1 and 4.

Claim 28 is dependent from Claim 24 and contains all the limitations of Claim 24. Claim 28 adds the same limitation to Claim 24 as Claim 5 adds to Claim 1. The Examiner has rejected Claim 28 for the same reasons as he rejected Claim 5. Therefore, the Applicants respectfully assert that the rejection of Claim 28 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 1 and 5.

Claim 32 is dependent from Claim 28 and contains all the limitations of Claim 28. Claim 32 adds the same limitation to Claim 28 as Claim 9 adds to Claim 5. The Examiner has rejected Claim 32 for the same reasons as he rejected Claim 9. Therefore, the Applicants respectfully assert that the rejection of Claim 32 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the same reasons as Claims 1, 5, and 9.

Claim 33 is an independent claim directed to a method of managing a memory device two steps. In step 1, the said memory device is partitioned into a plurality of memory spaces on an as-needed basis and in step 2 a memory attribute is associated with each memory space that determines a use of each of the memory spaces. The Examiner

states that *Kane* teaches the invention of Claim 33 and cites *Kane*, column 1, lines 56-59. In lines 56-59, *Kane* describes a portion of his invention including a memory control unit for a microprocessor that can run multiple tasks wherein the control unit has segment registers for directing memory access requests to specified segments in memory associated with the memory control unit. In particular, the microprocessor assigns the privilege levels to the segments in memory and assigns privilege levels to the multiple tasks as a form of memory protection. The Examiner is suggesting that the memory segments are analogous to the partitioned memory spaces of Claim 33. The Examiner then states that *Kane* teaches associating memory attributes with each memory space that determine a use of the memory spaces and cites *Kane*, column 1, lines 56-59. If the memory segments in this recitation are equivalent to the memory spaces of Claim 33, then *Kane* assigns <u>privilege levels</u> to the <u>memory spaces</u> which <u>do not determine a use of the memory spaces</u> but rather determine which tasks with matching privilege may use the memory spaces. This recitation of *Kane* <u>does not teach step 2</u> of Claim 33. Therefore, the Applicants respectfully assert that the rejection of Claim 33 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the reasons stated above.

Claim 34 is dependent from Claim 33 and contains all the limitations of Claim 33. Claim 34 adds the limitation that a particular memory attribute has a corresponding load/store instruction. The Examiner states that *Kane* teaches the invention of Claim 34 and cites *Kane* Fig. 1, element 114 and column 2, lines 33-37. Element 114 of Fig. 1 is an instruction queue which buffers instructions awaiting execution. The instruction queue 114 in no way indicates that a particular memory attribute has a corresponding load/store instruction. In *Kane*, load/store instructions are directed to loading or storing data from particular addresses in memory. *Kane* only teaches that the memory may have segments with various levels of privilege. Nowhere does *Kane* teach or describe a memory attribute having a corresponding load/store instruction. In column 2, lines 33-37, *Kane* provides a brief description of Fig. 5. Fig. 5 of *Kane* is illustrating segment registers and descriptor registers and how the descriptor registers relate to memory segments. However, nowhere in this recitation does *Kane* teach or describe a particular

memory attribute having a corresponding load/store instruction as recited in Claim 34 of the present invention. Therefore, the Applicants respectfully assert that the rejection of Claim 34 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the reasons stated above and for the same reasons as Claim 33.

Claim 35 is dependent from Claim 34 and contains all the limitations of Claim 34. Claim 34 adds the limitation that a load/store instruction associated with a first memory attribute causes an error condition if attempted on a memory space with a second memory attribute. The Examiner states that *Kane* teaches the invention of Claim 35 and cites *Kane*, column 1, lines 64-column 2, lines 8. In this recitation, *Kane* states that "The memory control unit further includes at least one programmable protection register accessible by the protection circuit. The programmable protection register is indexed by an index formed by combining a privilege level of the task requesting the specified memory segment and a privilege level of the specified memory segment. The protection register outputs at least one value responsive to the index that indicates whether the memory segment is accessible by the task. The protection circuit permits the memory control unit to load a segment register to identify the specified memory segment only when the specified memory segment is accessible by the task." *Kane* is describing how his memory control unit uses a programmable protection register and an index to generate and output a value indicating whether a memory segment is accessible by the task. *Kane* does not teach or suggest that memory attributes have corresponding load/store instructions nor does *Kane* teach that using a load/store instruction associated with a first memory attribute to access a memory with a second memory attribute will cause an error condition. Therefore, the Applicants respectfully assert that the rejection of Claim 35 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the reasons stated above and for the same reasons as Claim 34.

Claim 36 is dependent from Claim 33 and contains all the limitations of Claim 33. Claim 36 adds the limitation that each of the memory attributes are stored in a memory page table used to manage the memory device. The Examiner states that *Kane*

teaches the invention of Claim 36 and cites *Kane*, column 1, lines 64-66. In this recitation, *Kane* states that "the protection circuit verifies that a task requesting a specified memory segment is privileged to access the specified memory segment." Nowhere in this recitation does *Kane* teach or suggest that memory attributes, assigned to memory spaces and having corresponding load/store instructions are stored in a memory page table used by the memory device. Therefore, the Applicants respectfully assert that the rejection of Claim 36 under *35 U.S.C. §102(b)* as being anticipated by *Kane* is traversed for the reasons stated above and for the same reasons as Claim 33.

II. REJECTION UNDER 35 U.S.C. § 103(a)

The Examiner rejected Claims 6, 8, 14, 21, and 29-31 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of U.S. Patent No. 5,657,475 to *Gillespie et al.* (hereafter "*Gillespie*").

To establish a *prima facie* case of obviousness, the Examiner must meet three basic criteria. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be some reasonable expectation of success. Finally, the prior art reference must teach or suggest all the claim limitations.

Claim 6 is dependent from Claim 2 and contains all the limitations of Claim 2. Claim 6 adds the limitation that the stack memory load/store instructions are executed on a CPU comprising an IA64 architecture. The Examiner states that *Kane* teaches and describes a method for stack memory protection. While *Kane* does teach a method of memory protection using privilege levels, the Applicants have shown that *Kane* does not teach the method of stack memory protection recited in independent Claim 1 and dependent Claim 2. The Examiner further states that *Kane* does not explicitly teach that the stack memory load and store instructions are executed on a CPU comprising an IA64 architecture. However, the Examiner states that "*Gillespie* discloses stack memory load

and store instructions are executed in parallel (IA64 architecture) to the multi-ported registers." The Applicants respectfully assert that *Gillespie* cannot teach or suggest the IA64 architecture as it was not introduced by Intel® until after the year 1999 which is three years after the patent to *Gillespie* issued and five years after *Gillespie* filed his patent. Whether IA64 executes instruction in parallel is not relevant to the invention of Claim 6.

In the IA64 architecture, there is a memory stack and a register stack. The memory stack is the traditional stack used for procedure local data and to some degree for saving and restoring register contents. The register stack is a separate stack that the processor uses transparent to software to spill and fill to and from registers. An IA64 processor provides 128 general purpose registers for the programmer. However, registers r32-r128 are "stacked" registers. In actual implementation, there are some number of physical registers that are used to represent the programmable visible registers r32-r128. The processor manages the set of physical registers similar to a memory cache, in that, it can spill and fill the actual contents of these physical registers and can re-name the virtual to physical mappings of the registers. Because of this, the OS must provide two stack regions for every execution context. This complicates traditional "boundary" enforcement and stack placement (within the address space), to catch/trap cases where a program exceeds an allotted stack space. Traditionally, this is done using "red zones" which have problems explained in the Specification of the present invention and earlier in this response.

For these reasons, the Applicants respectfully assert that the rejection of Claim 6 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie* is traversed.

Claim 8 is dependent from Claim 7 and contains all the limitations of Claim 7. Claim 8 adds the limitation that the processor stack of Claim 7 is an IA64 register stack. Claim 7 is rejected by the Examiner under *35 U.S.C. §102(b)* as being anticipated by *Kane*. The Applicants have shown that *Kane* does not teach or disclose the invention of

Claim 7. The Examiner states that Claim 8 is rejected as applied above to Claim 7. Further, the Examiner states that *Gillespie* discloses a method for stack memory protection wherein the processor stack is an IA64 register stack and cites *Gillespie*, column 3, lines 50-63. Again, the Applicants respectfully assert that *Gillespie* cannot teach or suggest the IA64 architecture as it was not introduced by Intel® until after the year 1999 which is three years after the patent to *Gillespie* issued and five years after *Gillespie* filed his patent. Further, in the reference of *Gillespie* cited by the Examiner *Gillespie* does not teach or suggest the relevance of the IA64 architecture. Therefore, the Applicants respectfully assert that the rejection of Claim 8 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie* is traversed.

Claim 21 is dependent from Claim 17 and contains all the limitations of Claim 17. Claim 21 adds the limitation that the stack memory load and store instructions of Claim 17 are executed on a CPU comprising an IA64 architecture. The Examiner has rejected Claim 21 for the same reasons as he has rejected Claim 6. Claim 21 adds the same limitation to the data processing system of Claim 17 as Claim 6 adds to the method of Claim 2. Therefore, the Applicants respectfully assert that the rejection of Claim 21 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie* is traversed for the same reasons as Claim 6.

Claim 29 is dependent from Claim 25 and contains all the limitations of Claim 17. Claim 29 adds the limitation that the stack memory load and store instructions of Claim 25 are executed on a CPU comprising an IA64 architecture. The Examiner has rejected Claim 29 for the same reasons as he has rejected Claim 6. Claim 29 adds the same limitation to the data processing system of Claim 25 as Claim 6 adds to the method of Claim 2. Therefore, the Applicants respectfully assert that the rejection of Claim 29 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie* is traversed for the same reasons as Claim 6.

Claim 30 is dependent from Claim 25 and contains all the limitations of Claim 17. Claim 30 adds the limitation that first memory stack is a processor stack used by a

processor to load and store hardware register contents during program execution, wherein the processor stacks transparent to a programmer or a compiler. The Examiner has rejected Claim 30 for the same reasons as he has rejected Claim 29. Therefore, the Applicants respectfully assert that the rejection of Claim 30 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie* is traversed for the same reasons as Claim 29.

Claim 31 is dependent from Claim 30 and contains all the limitations of Claim 30. Claim 31 adds the limitation that the stack memory load and store instructions of Claim 30 are executed on a CPU comprising an IA64 architecture. The Examiner has rejected Claim 31 for the same reasons as he has rejected Claim 6. Claim 31 adds the same limitation to the computer program product of Claim 30 as Claim 6 adds to the method of Claim 2. Therefore, the Applicants respectfully assert that the rejection of Claim 31 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie* is traversed for the same reasons as Claim 6.

III. CONCLUSION

The Applicants have traversed the rejections of Claims 1-5, 7, 9-13, 15-21, 22-28, and 32-36 under *35 U.S.C. §102(b)* as being anticipated by *Kane*.

The Applicants have traversed the rejections of Claims 6, 8, 14, 21, and 29-31 under *35 U.S.C. §103(a)* as being unpatentable over *Kane* in view of *Gillespie*.

The Applicants, therefore, respectfully assert that Claims 1-32 are now in condition for allowance and request an early allowance of these claims.

Applicants respectfully request that the Examiner call Applicants' attorney at the below listed number if the Examiner believes that such a discussion would be helpful in resolving any remaining problems.

Respectfully submitted,

WINSTEAD SECHREST & MINICK P.C.

Patent Agent and Attorney for Applicants

By: Richard F. Frankeny

Richard F. Frankeny
Reg. No. 47,573
Kelly K. Kordzik
Reg. No. 36,571

P.O. Box 50784
Dallas, Texas 75201
(512) 370-2872

AUSTIN_1\251636\1
7047-P365US